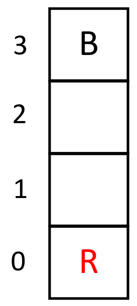
Written Question for Project 2.

Consider the game "First Queen" played on a 1 x 4 board by two players, Red (R), and Black (B). The initial state is as shown below. Red goes first. Each player is allowed to move to a vacant adjacent square (i.e., go either up or down one space). If R is blocked by B, and the square beyond B is vacant, R gets the option to move by jumping over B to that vacant square. B has a similar option under similar circumstances. If R makes it to row 3 that is a win for R, worth 10 points (and b gets -10). Similarly, B wins at row 0. Rather than actually becoming a queen, the game ends when either R gets to row 3 or B gets to row 0.



(i) Draw the full game tree with each state shown as an ordered pair (R's row, B's row). The initial state is therefore (0, 3). Whenever a state is reached that repeats one that's an ancestor, make the new instance be a leaf node and put a double circle around it.

(ii) Mark each node with a value. The double-circled nodes get the special value "?". The other leaf nodes represent end-of-game nodes and get as their value whatever Red gets in that game. All other nodes should get backed-up minimax values. Write the values in single circles. Explain how you processed "?" nodes, and your reasons for doing it that way.

(iii) Why would regular minimax fail on this tree, and how you could fix the algorithm to handle this sort of tree? Explain whether or not your new algorithm produces optimal results on any game tree with leaf nodes of the "?" type as here.

(iv) Suppose that instead of a 4-square board, we consider similar games having *n* squares where *n* > 2. Prove that R can always win when *n* is even, but B can always win if *n* is odd.